Macintoshモジュールリファレンス

リリース *2.3.3*

Guido van Rossum
Fred L. Drake, Jr., editor
日本語訳: Python ドキュメント翻訳プロジェクト

平成 16 年 6 月 28 日

PythonLabs
Email: docs@python.org

概 要

このライブラリリファレンスマニュアルでは、Macintosh 用の Python 拡張に関して詳しく記述します。*Python Library Reference* も同時に参照しながら利用するようにしてください。標準ライブラリと組み込み型はそちらに詳しく書かれています。

このマニュアルを読むにあたっては、Python 言語の基礎知識を持っていることが必要です。Python の肩のこらない入門編が必要なら、*Python Tutorial* を読んでください。*Python Reference Manual* は、構文や意味論に関する疑問を解決するため、それなりに分かった人が読むべきものです。最後にひとつ。インタプリタの拡張と組み込み *Extending and Embedding the Python Interpreter* という名のマニュアルでは、Python へ新たに拡張機能を追加する方法と、他のアプリケーションに組み込む方法について述べています。

目次

一章

Mac OS 9でPythonを利用する

Macintosh 上での Python の使い方は、特に Mac OS 9 上においては UNIX ライクなシステムや Windows システム上で使うのと全く異なっています (MacPython-OSX は完全な UNIX Python を含んでいるため UNIX ライクなシステムとほぼ同じ知識が利用できます) 。ほとんどの Python 文書や、“公式” 文書、出版された本のたぐいも、UNIX ライクなシステムや Windows システム上でどう Python を使うかについて述べているにすぎず、 Python を利用するのは MacPython-OS9 が初めてという人が混乱する原因になっています。この章では、簡単な入門編として Macintosh 上で Python を具体的にどう使うのかを解説していきます。

IDE についての章 (1.3章参照) は MacPython-OSX でも利用できます。

1.1 MacPython-OSX の取得とインストール

Python 2.3a2 の時点で、自分のマシンに MacPython-OSX をインストールする唯一確実な方法はソース配布物を取得し、"framework Python"と呼ばれるものを自分で構築する事です。構築方法の詳細は ‘Mac/OSX/README’ に記述されています。

バイナリインストーラーが利用可能になりしだい、詳細が `http://www.cwi.nl/~{}jack/macpython.html` に掲示されるでしょう。

多くのものがインストールされます。

- ‘Applications’ フォルダ中に ‘MacPython-2.3’ フォルダが作成されます。このフォルダ中に、PythonIDE 統合開発環境、ファインダからのダブルクリックで Python スクリプトを起動するための PythonLauncher、そして Package Manager が含まれています。
- 標準 UNIX コマンドライン Python インタプリタが ‘/usr/local/bin/python’ にインストールされます。しかし、‘/usr/local/lib/python’ は除きます。
- 全ての機能の実体として ‘/Library/Frameworks/Python.framework’ にフレームワークがインストールされます。しかし、普通はこの事を知っている必要はありません。

MacPython をアンインストールするには、単にこの 3 つを削除するだけです。

PythonIDE は、そのヘルプメニューからアクセスすることができる "MacPython Help" と呼ばれる Apple ヘルプビューア書類を含んでいます。Python にまったく不慣れなら、そのドキュメント中の IDE 入門を読み始めるのが良いでしょう。

他の UNIX プラットフォーム上の Python に精通しているなら、UNIX シェルからの Python スクリプトの実行についてのセクションを読むのが良いでしょう。

1.1.1 Python スクリプト実行方法

Mac OS X 上で Python を始めるには PythonIDE 統合開発環境に触れてみるのが最良の方法です。章 1.3 を見るか、IDE が起動しているならヘルプメニューから呼び出せる Apple ヘルプビューア書類の IDE 入門を

読みながら IDE に触れてみてください。

ターミナルウィンドウのコマンドライン、あるいはファインダから Python スクリプトを実行したければ、スクリプトを作成するために最初にエディタが必要になるでしょう。Mac OS X には、**vi** および **emacs** など多くの標準 UNIX コマンドラインエディタが付属します。もし Mac らしいエディタを利用したいなら Bare Bones Software (http://www.barebones.com/products/bbedit/index.shtml) の **BBEdit** か **TextWrangler** が良いでしょう。フリーウェアの **BBEdit Lite** の公式サポートは中止されましたが、まだ利用可能です。**AppleWorks** や他のワードプロセッサで ASCII としてファイルを保存できる物なら利用可能ですが、**TextEdit** はデフォルトでは '.rtf' フォーマットで保存してしまうので、そのままでは利用しない方がいいでしょう。

ターミナルウィンドウから自作のスクリプトを起動するにはシェルの検索パスに '/usr/local/bin' を '/usr/bin' よりも前に設定する必要があります。これを設定しないと Apple 提供の Python が起動されてしまいます。(Mac OS X 10.2.4 時点では Python 2.2 です)

ファインダから自作スクリプトを実行するには、2 種類方法があります。

- **PythonLauncher** にドラックする
- ファインダの情報ウィンドウで自作スクリプト (もしくはいずれかの.py スクリプト) の「このアプリケーションで開く」の設定のデフォルトを **PythonLauncher** にして、スクリプトをダブルクリックする。

PythonLauncher は、あなたのスクリプトをどのように起動するかコントロールするための様々な設定ができます。オプションを押しながらドラッグするとその実行においてのみ設定の変更がされます。また変更をずっと反映させるためには PythonLauncher の設定メニューから変更してください。

1.1.2 GUI を利用したスクリプトの実行

Mac OS X では Aqua ウィンドウマネージャに依存するプログラム (言いかえれば GUI を持っているものすべて) は特別な方法で起動する必要がある事を知っておく必要があります。GUI を持ったスクリプトを実行するには **python** のかわりに **pythonw** を利用してください。

1.1.3 設定

MacPython は、PYTHONPATH のような標準 UNIX 環境変数をすべて利用しますが、ファインダは自作の '.profile' や '.cshrc' を読まないので、ファインダから起動するプログラムのためにこれらの変数を設定することは非標準です。ファインダのためには '~/.MacOSX/environment.plist' ファイルを作成する必要があります。詳細に関しては Apple Technical Document QA1067 を見てください。

追加の Python パッケージのインストールは、パッケージマネージャによってかなり簡単にできます。詳細は MacPython ヘルプを見てください。

1.2 MacPython-OS9 の取得とインストール

最新リリースは、今後現われる実験的な新版と同様に、Jack Jansen により運営されている MacPython ページ:http://homepages.cwi.nl/\T1\textasciitildejack/macpython.html で見つかります。

使い方に関する最新情報は、配布物に含まれる 'README' を参照してください。

MacPython-OS9 は Mac OS X 上で正常に起動しますし、クラシック環境の中ではなくネイティブモードで起動します。それでも、CFM に基づいた Python を必要とする特別な理由がない限り、Mac OS X 上で MacPython-OSX を利用しない理由にはなりません。

1.2.1 対話型インタプリタを使う

Python ドキュメントで使われている対話型インタプリタは、**PythonInterpreter** アイコンをダブルクリックすると起動します。アイコンは 16 トンの重りが落ちてきたように見えるものになっています。[1]バージョン情報と '>>> ' プロンプトが表示されるはずです。これは標準ドキュメントに書かれているそのままに使えます。

1.2.2 Python スクリプトの走らせ方

既に手元にある Python スクリプトを走らせるにはいくつかの方法があります。普通は、"ドラッグ＆ドロップする" か "ダブルクリックする" か、この 2 つの方法のいずれかを使います。他にも、IDE(節 1.3 を参照) の中から走らせるとか、AppleScript 経由で起動する方法があります。

ドラッグ＆ドロップする

一番簡単に Python スクリプトを起動させるのは、"ドラッグ＆ドロップ" 経由で行うことです。これはファインダで、テキストファイルをワードプロセッサのアイコンの上に "ドラッグ" し、そこに "ドロップ" して起動させるのに似ています。"ドラッグ＆ドロップ" による Python スクリプトの起動には **PythonInterpreter** を使うように気をつけてください。**IDE** や **Idle** では、以下に示すのとは、異なる振る舞いになってしまいます。

ただ問題もいくつかあります。

- **PythonInterpreter** の上にスクリプトをドロップした後にウィンドウがフラッシュするが、その後消えてしまう場合。恐らく、これはシステムの設定関連の問題です。**PythonInterpreter** は実行完了の後すぐに終了するよう設定されているのに、スクリプトが何か表示するものがあったらしばらく待つ事を想定して作成されている場合です。これを修正するには、節 1.2.5 を参照してください。
- スクリプトのアイコンを **PythonInterpreter** の上に重ねようとふらふらさせても、**PythonInterpreter** がハイライトにならない場合。これは、たぶんクリエータコードと文書のタイプがセットされていない (あるいは誤ってセットされている) ためです。– こういうことは、ファイルを Mac でないコンピュータから持ってきた場合に時々おこることです。詳細は節 1.2.2 を参照してください。

クリエータをセットしてダブルクリックする

起動したいスクリプトが適当なクリエータコードとファイルタイプを持っているなら、スクリプトを起動するには単にダブルクリックすれば良い。"ダブルクリック可能" にするには、ファイルが 'TEXT' タイプで、クリエータコード '`Pyth`' である必要があります。

クリエータコードとファイルタイプをセットするには、IDE で行うか (節 1.3.2 と 1.3.4 を参照) Python モードを持つエディタ (**BBEdit** など) で行うか – これについては節 2 を参照、あるいは他の Mac のユーティリティで行えます。しかし、スクリプト ('fixfiletypes.py') が MacPython に同梱されていて、このスクリプトで Python により適切なタイプとクリエータコードをセットできるようになっています。

'fixfiletypes.py' スクリプトは、指定したディレクトリに対してファイルタイプとクリエータコードを変えます。'fixfiletypes.py' は以下のよう使います。

1. スクリプトを、MacPython 配布物の Mac フォルダの中にある scripts フォルダに置く。
2. 修正したいスクリプトを、全部ひとつのフォルダに入れる。

[1]訳注：なぜアイコンが 16 トンの重りなのかを疑問に思うかもしれません。これは Monty Python's Flying Circus に由来します。Monty Python's Flying Circus を見てみるとよいでしょう。

3. ‘fixfiletypes.py’ アイコンをダブルクリックする。
4. 修正したいファイルの入ったフォルダを選んで “Select current folder” ボタンを押す。

1.2.3 コマンドライン引数をシミュレートする

MacPython-OS9 でコマンドライン引数をシミュレートするには２つの方法があります。

1. インタプリタオプション経由
   - スクリプトを起動する時、オプションキーを押す。こうすると Python インタプリタオプションのダイアログボックスが現われます。
   - “Set UNIX-style command line..” ボタンをクリックする。
   - “Argument” フィールドに引数をタイプする。
   - “OK” をクリックする。
   - “Run” をクリックする。
2. ドロップ＆ドロップ経由。スクリプトをアプレットとして保存した場合には (節 1.3.4 を参照)、“ドラッグ＆ドロップ” 経由でコマンド引数もシミュレートできます。この場合、アプレットにドロップされたファイルの名前は `sys.argv` に追加され、スクリプトからはコマンドラインでタイプされたように見えます。UNIX システムでのように、`sys.srgv` の第一項目は、アプレットへのパスになり、残りの項目はアプレットにドロップされたファイルです。

1.2.4 Python スクリプトを作成する

Python スクリプトは単純なテキストファイルなので、テキストファイルが作れるならどのような方法でも作成可能です。しかし、他にも特徴を供えた専用ツールもあります。

エディタでスクリプトを作成

MSWord や **AppleWorks** などのワードプロセッサプログラムでテキストファイルを作成することもできますが、ファイルを “ASCII” か “プレインテキスト” で保存するよう気を付けてください。

Python モード付きのエディタ

テキストエディタのうちいくつかは、Python スクリプトを作成するための機能が付加されています。例えば、コードを読みやすくするために Python キーワードに色を付けたり、モジュールのブラウズ、組み込みデバッガなどの機能が含まれています。こうしたエディタには、**Alpha**、**Pepper**、 **BBedit**、MacPython IDE(節 1.3) があります。[2]

BBedit

BBEdit でスクリプトを作成する時は、保存したファイルをダブルクリックすると起動できるように、エディタで Python クリエータコードを与えると便利です。

- **BBEdit** を立ち上げる。
- “Preferences” を “Edit” メニューを選択する。

[2]訳注：**mi**(`http://www.mimikaki.net/`) にも Python モードがありますので利用してみてください。

- スクローリングリストから “File Types” を選択する。
- “Add...” ボタンをクリックし、ナヴィゲーションダイアログから MacPython 配布物のディレクトリから **PythonInterpreter** を選んで、“open” をクリックする。
- 設定パネルの “Save” ボタンをクリックする。

1.2.5 設定を行なう

MacPython 配布物には、作業を円滑に行なうために MacPython 環境をカスタマイズするのを助けるアプレット **EditPythonPrefs** が付属します。

EditPythonPrefs

EditPythonPrefs を利用して自分が望むような動作を Python にさせるための設定をする事ができます。**EditPythonPrefs** は、初期設定を設定することと、特別版の Python 実行ファイルをドロップしてカスタマイズすること、の 2 つの事ができます。例えば、普段は正常終了したら出力ウィンドウが閉じるが、正常終了した時に出力ウィンドウを開きっぱなしにしたいなどの理由で **PythonInterpreter** の別のコピーを持ちたい場合、後者の方法で手軽にできます。

デフォルト初期設定を変更するためには、単に **EditPythonPrefs** をダブルクリックしてください。インタープリタの 1 つのコピーのみに対する変更は、**EditPythonPrefs** に対象となるコピーをドロップしてください。さらに、**Python IDE** および任意のアプレットの初期設定を設定するために、この方法で **EditPythonPrefs** を使用することができます。節 1.3.4 を参照してください。

モジュール検索パスにモジュールを追加する

`import` 文を実行する時、Python は `sys.path` で定義した場所のモジュールを探します。Mac 上で `sys.path` を編集するには、**EditPythonPrefs** を起動し、パスを上部の大きめのフィールドに入力します (1 行に 1 パス)。

MacPython はメイン Python ディレクトリを定義しているので、フォルダをメイン Python ディレクトリに追加するのが最も簡単な方法です。作成したスクリプトを入れた “My Folder” という名前のフォルダをメイン Python ディレクトリに追加するには、‘`$(PYTHON):My Folder`’ を新規行として入力します。

OS 9 かそれ以前の OS でデスクトップを追加するには、‘`StartupDriveName:Desktop Folder`’ を新規行として入力します。

デフォルトの起動オプション

EditPythonPrefs ダイアログボックスの “Default startup options...” ボタンは、多くのオプションを提供します。例えば、スクリプトが終了した後で “Output” ウィンドウを開いたままにしたり、スクリプトの実行が終了した後で対話モードに入るようにしたりできます。後者は、スクリプトの途中で生成されたオブジェクトを検査する場合にとても便利です。

1.3 統合開発環境

Python IDE(統合開発環境) は独立したアプリケーションで、Python コードのテキストエディタや、クラスブラウザ、グラフィカルデバッガなどとして動作します。

1.3.1 “Python Interactive” ウィンドウを使う

“ドラッグ＆ドロップ” の手法を使えない以外は、このウィンドウは **PythonInterpreter** と同様に使えます。通常のドラッグ＆ドロップの代わりに、スクリプトを **Python IDE** アイコンの上にドロップすると、別々なスクリプトウィンドウでファイルが開かれます (こうすると手動で実行させることができます – 節 1.3.3を参照)。

1.3.2 Python スクリプトを書く

Python IDE は、対話的に使うだけでなく、Python プログラムを書き上げ、順次保存したり、全体や一部分を実行したりすることもできます。

新たにスクリプトを作成したり、前に保存したスクリプトを開いたり、“File” メニューの適当なメニューアイテムを選択することで、現在開いているスクリプトを保存することもできます。 Python スクリプトを **Python IDE** の上にドロップすると、ファイルが開いて編集可能になります。

スクリプトを **Python IDE** で開きたいが “Open” ダイアログボックスから場所を特定できない時や、“Can’t open file of type ...” のようなエラーメッセージが出る時は、節 1.2.2 を参照してください。

Python IDE がスクリプトを保存する時に使われるクリエータコードの設定は、ドキュメントウィンドウの一番右上の小さな黒い三角形をクリックすると見つかり、“save options” を選択することで設定できます。デフォルトではファイルのクリエータコードは **Python IDE** になります。つまり保存されたファイルのアイコンダブルクリックするとそのまま IDE で編集できます。**PythonInterpreter** で開いて実行するように、この動作を変えたい時もあるでしょう。そうするには単に “save options” から “Python Interpreter” を選べば良いだけです。こうしたオプションは**ファイル単位**の設定であり、アプリケーション単位の設定でないことに注意してください。

1.3.3 統合開発環境の中からスクリプトを実行する

Python IDE の最前面のウィンドウで全部実行 (run all) ボタンを押すと、そのウィンドウのスクリプトを実行できます。しかし、もし Python の習慣通りに ‘`if __name__ == "__main__":`’ と書いても、スクリプトはデフォルトでは “__main__” に **ならない** ことを知っておいてください。そういう風に動作させるには、ドキュメントウィンドウの一番右上の小さな黒い三角形から、“Run as __main__” オプションを選ばなくてはいけません。こうしたオプションは **ファイル単位** の設定であって、アプリケーション単位でないことに注意してください。ただし、このオプションは、保存した後でも **そのまま** でありつづけます。これをオフにするには、単にもう一度選択すればよいだけです。

1.3.4 “Save as” 対 “Save as Applet”

Python スクリプトを書き終わったら、“applet” として保存することもできます (“File” メニューの “Save as applet” を選択する)。こうする重要なメリットは、できあがったアプレットの上にファイルやフォルダをドロップすると、スクリプトに対してコマンドライン風に引数として渡すことができる点にあります。ただし、アプレットを別ファイルとして保存するように気をつけて、くれぐれもせっかく作ったスクリプトを上書きしないようにしてください。そうしてしまうと、再度編集できなくなってしまいます。

“ドラッグ＆ドロップ” 経由でアプレットに渡された項目にアクセスするためには、標準的な `sys.argv` の動作を使えばよいです。詳しい説明は Python の標準ドキュメントを見てください。

スクリプトをアプレットとして保存しても、 Python がインストールされていないシステムでは実行できないことに注意してください。

二章

MacPython モジュール

このドキュメントで記述されている次のモジュールは、いずれも Macintosh でのみ利用可能です。

mac	os モジュールの実装
macpath	MacOS のパス操作関数
macfs	FSSpec、エイリアスマネージャ、**finder** エイリアス、標準ファイルパッケージのサポート。
ic	インターネット設定へのアクセス。
MacOS	Mac OS 固有のインタープリタ機能へのアクセス。
macostools	ファイル操作に便利なルーチン。
findertools	**finder** の Apple Events インターフェースのラッパー。
EasyDialogs	基本的な Macintosh ダイアログ。
FrameWork	対話型アプリケーション・フレームワーク
autoGIL	イベントループ中のグローバルインタープリタの取り扱い

2.1 mac — os モジュールの実装

このモジュールは、標準モジュール os でサポートされるのと同様の機能を、Mac OS 9 オペレーティングシステムに依存した機能として実装しています。このモジュールを利用する一番良い使い方は posix モジュール経由で使うことです。このモジュールは MacPython-OS9 にのみ存在し、MacPython-OSX 上では posix を利用します。

このモジュールでは次の関数が使えます。chdir()、close()、dup()、fdopen()、getcwd()、lseek()、listdir()、mkdir()、open()、read()、rename()、rmdir()、stat()、sync()、unlink()、write()、そして例外 error も定義されています。stat() により返される時間は浮動小数点数で、 MacPython-OS9 で使われる他の時間の値と同様です。

2.2 macpath — MacOS のパス操作関数

このモジュールは os.path モジュールの Macintosh 実装です。このモジュールで、os.path への最も汎用性のあるアクセスができます。os.path のドキュメントに関しては、*Python Library Reference* を参照してください。

次の関数がこのモジュールで利用できます。normcase()、normpath()、isabs()、join()、split()、isdir()、isfile()、walk()、exists()。os.path で利用できる他の関数については、ダミーの関数として相当する物が利用できます。

2.3 macfs — 各種のファイルシステムのサービス

リリース 2.3 以降で撤廃された仕様です。 macfs モジュールは旧式のものです。FSSpec、FSRef、Alias の

代わりに Carbon.File あるいは Carbon.Folder を使用してください。ファイルダイアログには EasyDialogs を使用してください。

このモジュールは Macintosh FSSpec の操作、エイリアスマネージャ、**finder** エイリアス、標準ファイルパッケージへのアクセスを提供します。

関数やメソッドが *file* 引数をとる所では、引数は次の 3 つのうちのどれか 1 つです。つまり、(1) Macintosh のフルパス名あるいは部分パス名、(2) FSSpec オブジェクト、(3) *Inside Macintosh: Files* に記述された 3 要素のタプル (*wdRefNum*, *parID*, *name*)。それを含むフォルダが存在する限り、FSSpec は存在しないファイルを示すこともできます。MacPython ではパス名も同じように使用できますが、パス名と FSRefs の働きの違いのため、unix-Python では異なります。詳しくは Apple の文書を参照してください。

エイリアスと標準ファイルパッケージの詳細についてもその文書に書かれています。

FSSpec(*file*)
指定したファイルに対する FSSpec オブジェクトを作成します。

RawFSSpec(*data*)
文字列として FSSpec の C 構造体の生データを与えると FSSpec オブジェクトを作成します。主に FSSpec 構造体をネットワーク経由で得る場合に便利です。

RawAlias(*data*)
文字列としてエイリアスの C 構造体を生データとして与えると Alias オブジェクトを作成します。主に FSSpec 構造体をネットワーク経由で得る場合に便利です。

FInfo()
ゼロで埋めた FInfo オブジェクトを作成します。

ResolveAliasFile(*file*)
エイリアスファイルを解決します (オリジナルのファイルとの対応を取ります)。3 要素のタプル (*fsspec*, *isfolder*, *aliased*) を返します。ここで *fsspec* は解決後の FSSpec オブジェクト、もし *f sspec* がフォルダを指していたら *isfolder* は true、もしエイリアスだったら *aliased* は true(そうでなければファイルそのものの FSSpec オブジェクトが返されます) です。

StandardGetFile([*type, ...*])
ユーザに標準的な “入力ファイルを開く” ダイアログを提示します。オプションとして、4 つまでの 4 文字のファイルタイプを渡すことができて、ユーザーが選ぶファイルを制限することができます。この関数は FSSpec オブジェクトと、ユーザがキャンセルしないでダイアログを完了したかを示すフラグを返します。

PromptGetFile(*prompt*[, *type, ...*])
StandardGetFile() とほぼ同じですが、ダイアログの一番上に表示されるプロンプトを指定できます。

StandardPutFile(*prompt*[, *default*])
ユーザに標準的な “出力ファイルを開く” ダイアログを提示します。*prompt* はプロンプト文字列で、省略可能な *default* 引数で出力ファイル名の初期値を指定します。この関数は FSSpec オブジェクトと、ユーザがキャンセルしないでダイアログを完了したかを示すフラグを返します。

GetDirectory([*prompt*])
ユーザに非標準的な “ディレクトリを選ぶ” ダイアログを提示します。“select current directory” ボタンをクリックするときには、選んだディレクトリを開いていなくてはなりません。*prompt* はプロンプト文字列で、ダイアログの一番上に表示されます。FSSpec オブジェクトと、ユーザがキャンセルしないでダイアログを完了したかを示すフラグを返します。

SetFolder([*fsspec*])
ファイル選択ダイアログを提示する時に、最初に表示されるフォルダをセットします。*fsspec* で指定

するのはそのフォルダにあるファイルであって、フォルダそのものではありません (指定されたファイルは存在していなくても構いません)。もし引数を渡さない場合は、フォルダはカレントディレクトリ、つまり `os.getcwd()` で返されるディレクトリにセットされます。

システム 7.5 から “一般設定” コントロールパネルで、ユーザが標準ファイルパッケージの振る舞いを変えられることに注意してください。その設定しだいでこの関数の呼びだしは効果がなくなります。

`FindFolder`(*where, which, create*)
MacOS が管理する “特別な” フォルダ、例えばゴミ箱や初期設定フォルダなどの位置を探しだします。*where* は検索するディスク、*which* は探し出すフォルダを指定する 4 文字の文字列です。*create* をセットすると、そのフォルダが存在しなければ新たに生成します。(*vrefnum*, *dirid*) のタプルを返します。

where と *which* に与える定数は、標準モジュール *Carbon.Folders* にあります。

`NewAliasMinimalFromFullPath`(*pathname*)
与えられたファイルを指す最短の `alias` オブジェクトを返します。ファイルはフルパス名で与えなければなりません。これは存在しないファイルを指す `Alias` を作成する唯一の方法です。

`FindApplication`(*creator*)
4 文字のクリエータコード *creator* を持ったアプリケーションの位置を探し出します。この関数はアプリケーションを指す `FSSpec` オブジェクトを返します。

2.3.1 FSSpec オブジェクト

`data`
FSSpec オブジェクトの生データで、例えば他のアプリケーションに渡す場合に適しています。

`as_pathname`()
`FSSpec` で記述されたファイルのフルパス名を返します。

`as_tuple`()
`FSSpec` オブジェクトで記述されたファイルの (*wdRefNum*, *parID*, *name*) のタプルを返します。

`NewAlias`([*file*])
この FSSpec で記述されたファイルのエイリアスオブジェクトを作成します。省略可能な *file* パラメータを渡すと、エイリアスはそのファイルに対する相対指定で作成され、その他の場合は絶対指定となります。

`NewAliasMinimal`()
このファイルを指す最短エイリアスを作成します。

`GetCreatorType`()
このファイルの 4 文字のクリエータとタイプを返します。

`SetCreatorType`(*creator, type*)
このファイルに 4 文字のクリエータとタイプを設定します。

`GetFInfo`()
ファイルのファインダ情報を示す `FInfo` オブジェクトを返します。

`SetFInfo`(*finfo*)
ファイルのファインダ情報を *finfo*(`FInfo` オブジェクト) で与えた値に設定します。

`GetDates`()
作成日、修正日、バックアップ日を意味する 3 つの浮動小数点数からなるタプルを返します。

`SetDates`(*crdate, moddate, backupdate*)
ファイルの作成日、修正日、バックアップ日を設定します。各々の値は Python システム全体で標準的な浮動小数点数フォーマットです。

2.3.2 エイリアスオブジェクト

data
Alias レコードの生データで、リソースに書き込む時や他のプログラムに転送する時に適しています。

Resolve([*file*])
エイリアスを解決します。もしエイリアスが相対エイリアスとして作成されていた場合は、相対指定の起点となるファイルを渡す必要があります。指定先のファイルの FSSpec と、`Alias` オブジェクト自体が検索途中で修正されたかどうかのフラグを返します。もしファイルが存在しなくてそのファイルまでのパスが存在すれば、正しい FSSpec が返されます。

GetInfo(*num*)
C のルーチン `GetAliasInfo()` へのインターフェース。

Update(*file*[, *file2*])
与えられた *file* を指すエイリアスを更新します。もし *file2* 引数があれば、相対エイリアスが作成されます。

今のところ、リソースを `Alias` オブジェクトとして直接操作することは出来ません。そのため、`Update()` を呼んだ後か、`Resolve()` でエイリアスに変更があったと分かった後は、Python プログラムが責任もって `Alias` オブジェクトから `data` の値を取りだして、リソースを修正する必要があります。

2.3.3 FInfo オブジェクト

各々のフィールドが何を意味するかは *Inside Macintosh: Files* に完全に記述されているので、そちらを参照してください。

Creator
ファイルの 4 文字のクリエータコード。

Type
ファイルの 4 文字のタイプコード。

Flags
ファイルのファインダフラグを 16 ビットの整数で表現したもの。*Flags* のビット値は、標準モジュール `MACFS` で定義されています。

Location
ファイルのアイコンがフォルダの中で配置される位置。

Fldr
ファイルがあるフォルダ (整数表現)。

2.4 ic — インターネット設定へのアクセス

このモジュールは Macintosh のインターネット設定パッケージへのアクセスを提供します。このパッケージにはインターネットプログラムの設定、例えばメールアドレス、デフォルトのホームページなどが保存されています。それ以外にもインターネット設定は Macintosh のクリエータ／タイプとファイル名の拡張子との対応付けや、ファイルの転送方法 (バイナリ、アスキーなど) に関する情報を含んでいます。MacOS 9 以降では、このモジュールは “インターネット” という名前のコントロールパネルになりました。

`icglue` という低レベルの関連モジュールがあって、このモジュールはインターネット設定の基本的なアクセス機能を提供しています。この低レベルモジュールは文書化されていませんが、ルーチンの docstring でパラメータが説明されていますし、ルーチン名はインターネット設定にある Pascal や C の API と同じなので、このモジュールを使う場合は IC プログラマーのための標準的な文書が利用できます。

ic モジュールは例外 error と、インターネット設定から生じる全てのエラーコードに対するシンボル名を定義しています。詳細はソースを参照してください。

exception error
ic モジュール内部のエラーで発生した例外。

ic モジュールは以下のクラスと関数を定義しています：

class IC([*signature*[, *ic*]])
インターネット設定オブジェクトを作成します。*signature* は、現在のアプリケーションの 4 文字のクリエータ (デフォルトは'Pyth') で、他の IC の設定に影響する可能性があります。オプションの引数 *ic* はあらかじめ作成された低レベルの icglue.icinstance で、別の設定ファイルなどから設定を得る場合に便利です。

launchurl(*url*[, *hint*])
parseurl(*data*[, *start*[, *end*[, *hint*]]])
mapfile(*file*)
maptypecreator(*type, creator*[, *filename*])
settypecreator(*file*)
これらの関数は、後述する同名のメソッドへの “ショートカット” です。

2.4.1 IC オブジェクト

IC オブジェクトは対応付けのインターフェースを持っているので、メールアドレスは単に *ic*['MailAddress'] で得ることができます。値の代入も機能しますし、設定ファイルのオプションを変えることができます。

このモジュールは各種のデータ型を知っているので、IC 内部の表現を “論理的” な Python データ構造に変換します。ic モジュールをスタンドアロンで実行するとテストプログラムが実行され、IC データベースにある全てのキーと値のペアをリスト表示するので、文書代わりになります。

もしこのモジュールがデータをどう表わすか分からない場合は、data 属性に生のデータを入れた ICOpaqueData タイプのインスタンスを返します。このタイプのオブジェクトは値のアクセスだけでなく代入も可能です。

ディクショナリのインターフェースだけでなく、IC には以下のメソッドがあります。

launchurl(*url*[, *hint*])
与えられた URL を解析し、正しいアプリケーションを起動して URL を渡します。省略可能な *hint* は、'mailto:' などのスキーム名で、不完全な URL はこのスキームにあわせて完全化されます。もし *hint* が与えられていない場合は、不完全な URL は無効になります。

parseurl(*data*[, *start*[, *end*[, *hint*]]])
data の中から URL を検索して、URL の開始位置、終了位置、URL そのものを返します。省略可能な *start* と *end* で検索範囲を制限することができます。例えば、ユーザーが長いテキストフィールドをクリックしたとしても、テキストフィールド全体とクリック位置 *start* を渡すことができて、このルーチンによってユーザーがクリックしたテキストから URL 全体が返されます。既に述べたように、*hint* はオプションのスキームで、不完全な URL を完全化するために使われます。

mapfile(*file*)
与えられた *file* に対するマッピングエントリーを返します。*file* としてファイル名か macfs.FSSpec() の結果を渡すことができて、存在しないファイルであってもかまいません。

マッピングエントリーはタプル (*version*, *type*, *creator*, *postcreator*, *flags*, *extension*, *appname*, *postappname*, *mimetype*, *entryname*) として返されます。ここで *version* はエントリーのバージョン番号、*type* は 4 文字のファイルタイプ、*creator* は 4 文字のクリエータタイプ、*postcreator* は省略可

能な後処理用アプリケーションを示す 4 文字のクリエータで、このアプリケーションはファイルをダウンロードした後にファイルを後処理します。*flags* は各種のビットであり、転送をバイナリで行うかアスキーで行うかなどを表わします。*extension* はこのファイルタイプに対するファイル名の拡張子、*appname* はファイルが属するアプリケーションの出力可能な名前、*postappname* は後処理用アプリケーション、*mimetype* はこのファイルの MIME タイプ、最後の *entryname* はこのエントリーの名前です。

maptypecreator(*type, creator*[, *filename*])

与えられた 4 文字の *type* と *creator* を持つファイルに対するマッピングエントリーを返します。正しいエントリーが見つかりやすいように、オプションの *filename* を指定することもできます (例えばクリエータ`'????'` の場合など)。

マッピングエントリーは *mapfile* と同じフォーマットで返されます。

settypecreator(*file*)

存在する *file* をファイル名か `macfs.FSSpec()` の結果として与えると、拡張子から判断してクリエータとタイプを適切にセットします。ファインダにこの変更が伝えられるので、ファインダのアイコンは即座に更新されます。

2.5 MacOS — Mac OS インタプリタ機能へのアクセス

このモジュールは、Python インタプリタ内の MacOS 固有の機能に対するアクセスを提供します。例えば、インタプリタのイベントループ関数などです。十分注意して利用してください。

モジュール名が大文字で始まることに注意してください。これは昔からの約束です。

runtimemodel

`'carbon'` か `'macho'` のいずれかです。現在利用している Python が Mac OS X および Mac OS 9 互換性をもつ CarbonLib 形式、あるいは Mac OS X のみの Mach-O 形式をのどちらであるか判断できます。Python の初期のバージョンでは、値がさらに古い Mac OS 8 ランタイムモデル用の`'ppc'` である事があります。

linkmodel

インタープリタがどのような方法でリンクされているかを返します。拡張モジュールがリンクモデル間で非互換性かもしれない場合、パッケージはより多くの適切なエラーメッセージを伝えるためにこの情報を使用することができます。値は静的リンクした Python は `'static'`、Mac OS X framework で構築した Python は `'framework'`、標準の unix 共有ライブラリ (shared library) で構築された Python は `'shared'`、Mac OS 9 互換 Python では `'cfm'` となります。

exception Error

MacOS でエラーがあると、このモジュールの関数か、Mac 固有なツールボックスインターフェースモジュールから、この例外が生成されます。引数は、整数エラーコード (`OSErr` 値) とテキストで記述されたエラーコードです。分かっている全てのエラーコードのシンボル名は、標準モジュール `macerrors` で定義されています。

SetEventHandler(*handler*)

内部のインタプリタループでは、`ScheduleParams()` で止めないかぎり、Python は時々イベントをチェックします。イベントがある場合は、この関数を使うと、Python イベントハンドラ関数を渡せます。イベントはパラメータとして渡され、イベントが完全に処理された場合は、ハンドラ関数は非ゼロを返さなくてはなりません。それ以外はイベント処理は継続されます (例えば、イベントをコンソールウィンドウパッケージ渡すなど)。

イベントハンドラをクリアするには、パラメータなしで `SetEventHandler()` を呼び出します。既

にイベントハンドラがセットされているのに、さらにセットしようとするとエラーになります。

有効性：MacPython-OS9

SchedParams([*doint*[, *evtmask*[, *besocial*[, *interval*[, *bgyield*]]]]])

これはインタプリタの内部ループイベントハンドラに影響を与えます。*Interval* は、インタプリタがどれだけの頻度 (浮動小数点数の秒で表わされる) でイベント処理コードに入るかを指定します。真なら *doint* は割り込み (コマンドドット) チェックが行われます。*evtmask* はインタプリタに、イベントをマスクして (再描画、他のアプリケーションに切り替わるマウスクリックなど) イベント処理するよう指示します。*besocial* フラグは、他のプロセスが動作するチャンスを与えます。Python が最前面で動いている時は、最小限の実行時間が割り当てられ、Python が背景にある場合は、*interval* 当りに *bgyield* 秒が与えられます。

全てのパラメータはオプションで、現在の値がデフォルト値となります。この関数で返される値は、これらのオプションの既存の値からなるタプルです。デフォルトの初期値は、全ての処理がオンで、チェックは 1/4 秒毎、バックグラウンドで動作している場合はプロセッサも 1/4 秒毎に割り当てられます。

最も一般的な使用ケースは完全にインタープリタメインループ中の処理をできなくするために `SchedParams(0, 0)` を呼ぶことです。

有効性：MacPython-OS9

HandleEvent(*ev*)

イベントレコード *ev* を Python のイベントループに渡す、というよりは、`sys.stdout` ウィンドウ (Python をビルドしたコンパイラにもとづいて) のハンドラに渡されることになります。こうすると、Python プログラムが独自のイベント処理を行え、コマンドピリオドやウィンドウの切り替えが行えます。

`SetEventHandler()` でセットしたイベントハンドラからこの関数を呼びだそうとすると、例外が生じます。

有効性：MacPython-OS9

GetErrorString(*errno*)

MacOS のエラーコード *errno* のテキスト表現を返します。

splash(*resid*)

この関数は、*resid* で与えた DLOG リソースの内容で、スプラッシュウィンドウを画面に表示します。引数なしで呼びだすと、スプラッシュ画面を取り除きます。拡張モジュールをたくさんロードさせる前に、初期化のタイミングでアプレットにスプラッシュ画面を表示させたいときに、この関数が便利でしょう。

有効性：MacPython-OS9

DebugStr(*message* [, *object*])

Mac OS 9 上では、メッセージ *message* を出してローレベルデバッガに入ります。オプションの *object* 引数は使われませんが、デバッガから内容を容易に検査することができます。Mac OS X 上では 文字列が単に stderr に印字されます。

この関数を使うときは十分気を付けてください。MacsBug などのローレベルデバッガがインストールされていない場合は、システムがクラッシュしてしまいます。この関数は主に Python 拡張モジュールの開発者のために用意されています。

SysBeep()

ベルを鳴らします。

GetTicks()

システム起動時からのチック数 (clock ticks、1/60 秒) を得ます。

GetCreatorAndType(*file*)
2 つの 4 文字の文字列としてファイルクリエータおよびファイルタイプを返します。*file* 引数はパスもしくは、FSSpec、FSRef オブジェクトを与える事ができます。

SetCreatorAndType(*file, creator, type*)
ファイルクリエータおよびファイルタイプを設定します。*file* 引数はパスもしくは、FSSpec、FSRef オブジェクトを与える事ができます。*creator* と *type* は 4 文字の文字列が必要です。

openrf(*name* [, *mode*])
ファイルのリソースフォークを開きます。引数は組み込み関数 open() と同じです。返されたオブジェクトはファイルのように見えるかもしれませんが、これは Python のファイルオブジェクトではありませんので扱いに微妙な違いがあります。

WMAvailable()
現在のプロセスが動作しているウィンドウマネージャにアクセスします。例えば、Mac OS X サーバー上、あるいは SSH でログインしている、もしくは現在のインタープリタがフルブローンアプリケーションバンドル (fullblown application bundle) から起動されていない場合などのような、ウィンドウマネージャが存在しない場合は False を返します。

Mac OS 9 上ではこの関数はつねに True を返します。

2.6 macostools — ファイル操作に便利なルーチン

このモジュールには、Macintosh 上のファイル操作のための便利なルーチンがいくつか含まれています。ファイルのパラメータは全てパス名、あるいは FSRef か FSSpec オブジェクトで指定します。

macostools モジュールには以下の関数が定義されています。

copy(*src, dst*[, *createpath*[, *copytimes*]])
ファイル *src* を *dst* へコピーします。*createpath* が非ゼロなら、必要に応じて *dst* に至るまでのフォルダを作成します。このメソッドはデータとリソースフォークとファインダ情報をいくつか (クリエータ、タイプ、フラグ) をコピーします。オプションで作成日、修正日、バックアップ日の情報もコピーします (デフォルトでこれらもコピーします)。カスタムアイコン、コメント、アイコン位置はコピーされません。

copytree(*src, dst*)
必要に応じてフォルダを作成しながら、*src* から *dst* へ再帰的にファイルのツリーをコピーします。*src* と *dst* はパス名で指定しなければなりません。

mkalias(*src, dst*)
src を示すファインダエイリアス *dst* を作成します。

touched(*dst*)
ファイル *dst* のクリエータやタイプなどのファインダ情報が変わったことをファインダに知らせます。ファイルはパス名か FSSpec で指定できます。この呼び出しによってアイコンを再描画するようファインダに指示します。

BUFSIZ
copy に用いるバッファサイズで、デフォルトは 1 メガバイトです。

ファインダエイリアスの作成プロセスは、Apple の文書で明らかにされていません。そのため、mkalias() で作成されたエイリアスは互換性のない振る舞いをすることもあることに注意してください。

2.7 findertools — **finder** の Apple Events インターフェース

このモジュールのルーチンを使うと、Python プログラムからファインダが持ついくつかの機能へアクセスできます。これらの機能はファインダへの AppleEvent インターフェースのラッパーとして実装されています。全てのファイルとフォルダのパラメータは、フルパス名、あるいは FSRef か FSSpec オブジェクトで指定できます。

findertools モジュールは以下の関数を定義しています。

launch(*file*)
ファインダに *file* を起動するように命令します。起動が意味するものは *file* に依存します。アプリケーションなら起動しますし、フォルダなら開かれ、文書なら適切なアプリケーションで開かれます。

Print(*file*)
ファインダにファイルを印刷するよう命令します。実際の動作はファイルを選択し、ファインダのファイルメニューから印刷コマンドを使うのと同じです。

copy(*file, destdir*)
ファインダにファイルかフォルダである *file* をフォルダ *destdir* にコピーするよう命令します。この関数は新しいファイルを示す Alias オブジェクトを返します。

move(*file, destdir*)
ファインダにファイルかフォルダである *file* をフォルダ *destdir* に移動するように命令します。この関数は新しいファイルを示す Alias オブジェクトを返します。

sleep()
マシンがサポートしていれば、ファインダに Macintosh をスリープさせるよう命令します。

restart()
ファインダに、マシンが適切にリスタートするよう命令します。

shutdown()
ファインダに、マシンが適切にシャットダウンするよう命令します。

2.8 EasyDialogs — 基本的な Macintosh ダイアログ

EasyDialogs モジュールは Macintosh の単純なダイアログを含んでいます。全てのルーチンは省略可能なリソース ID のパラメータ *id* をとり、対応するダイアログアイテム（タイプとアイテム番号）があれば、この *id* によってダイアログ用の DLOG リソースを上書きできます。詳細についてはソースを参照してください。

EasyDialogs モジュールには以下の関数が定義されています。

Message(*str*[, *id*[, *ok=None*]])
メッセージテキスト *str* 付きのモーダルダイアログを表示します。テキストの長さは最大 255 文字です。ボタンのテキストはデフォルトでは “OK” ですが、文字列の引数 *ok* が与えられると、その文字列に設定できます。ユーザが “OK” ボタンをクリックすると、ユーザにコントロールが渡されます。

AskString(*prompt*[, *default*[, *id*[, *ok*[, *cancel*]]]])
ユーザに文字列値を入力するよう要求するモーダルダイアログ表示します。*prompt* はプロンプトメッセージで、省略可能な *default* 引数は入力文字列の初期値です（指定されないなら""が使われます）。“OK” と “Cancel” ボタンの文字列は *ok* と *cancel* の引数で変えることができます。文字列の長さは全て最大 255 文字です。AskString() は入力された文字列を返し、ユーザがキャンセルした場合は None を返します。

AskPassword(*prompt*[, *default*[, *id*[, *ok*[, *cancel*]]]])

ユーザに文字列を入力するように要求するモーダルダイアログを表示します。`AskString()` に似ていますが、テキストは点で表示されます。引数は `AskString()` のものと同じ意味です。

`AskYesNoCancel`(*question*[, *default*[, *yes*[, *no*[, *cancel*[, *id*]]]]])

プロンプト *question* と “Yes”、“No”、“Cancel” というラベルの 3 つボタンが付いたダイアログを表示します。“Yes” を押したら `1`、“No” なら `0`、“Cancel” なら `-1` を返します。`RETURN` キーを押した場合は *default* の値（*default* が設定されていないなら `0`）を返します。ボタンのテキストは *yes*、*no*、*cancel* の引数で変更できます。ボタンを表示させないためにはその引数に`""`を与えます。

`ProgressBar`([*title*[, *maxval*[, *label*[, *id*]]]])

プログレスバー付きのモードレスダイアログを表示します。これは後述の `ProgressBar` クラスのためのコンストラクタです。*title* は表示されるテキスト文字列（デフォルトは “Working...”）で、*maxval* は進行が完了するときの値（デフォルトは `0` で、不定量の作業が残っていることを示します）、*label* はプログレスバーの上に表示されるテキストです。

`GetArgv`([*optionlist*[*commandlist*[, *addoldfile*[, *addnewfile*[, *addfolder*[, *id*]]]]]])

コマンドライン引数のリストを作るのを補助するダイアログを表示します。引数を `getopt.getopt()` に渡すのに適した `sys.argv` のフォーマットのリストを返します。*addoldfile*、*addnewfile*、*addfolder* の引数はブーリアン型の引数です。これらが非ゼロなら、存在するファイル、まだ（おそらく）存在しないファイル、フォルダをコマンドラインのパスとしてそれぞれ設定できます。（注意:`getopt.getopt()` で認識されるために、オブションの引数はコマンドラインの中でファイルとフォルダの引数の前に示されなければなりません。）空白を含む引数は、空白をシングルクォートあるいはダブルクォートで囲んで指定できます。

ユーザが “Cancel” ボタンを押したら `SystemExit` 例外が発生します。

optionlist はポップアップメニューを示すリストで、ユーザはその中から選択できます。ポップアップメニューのアイテムは次の 2 つの形のうちの 1 つです。*optstr* あるいは `(`*optstr*`, `*descr*`)`。*descr* は短い説明の文字列で、これがあるとポップアップメニューから選択されている間、ダイアログに表示されます。*optstr* とコマンドライン引数の対応は以下の通りです。

***optstr* format**	**Command-line format**
`x`	**-x** （短いオプション）
`x:` あるいは `x=`	**-x** （値を持つ短いオプション）
`xyz`	**--xyz** （長いオプション）
`xyz:` あるいは `xyz=`	**--xyz** （値を持つ長いオプション）

commandlist は *cmdstr* あるいは `(`*cmdstr*`, `*descr*`)` の形のアイテムからなるリストで、*descr* は前述のように働きます。*cmdstr* はポップアップメニューに表示されます。選択されると *cmdstr* がコマンドラインに追加されますが、それに続く ‘`:`’ あるいは ‘`=`’ は（もし存在するなら）取り除かれます。

2.0 で追加された仕様です。

`AskFileForOpen`([*message*] [, *typeList*] [, *defaultLocation*] [, *defaultOptionFlags*] [, *location*] [, *clientName*] [, *windowTitle*] [, *actionButtonLabel*] [, *cancelButtonLabel*] [, *preferenceKey*] [, *popupExtension*] [, *eventProc*] [, *previewProc*] [, *filterProc*] [, *wanted*])

開くファイルをユーザに尋ねるダイアログを表示して、選択されたファイルあるいはユーザがキャンセルしたら *None* を返します。*message* は表示するテキストメッセージで、*typeList* は選択可能にする 4 文字のファイルタイプのリスト、*defaultLocation* は最初に表示するルフォルダのパスネームで FSSpec あるいは FSRef で指定します。*location* はダイアログを表示するスクリーン上の位置の `(x, y)` で、*actionButtonLabel* は OK ボタンの位置に “Open” の代わりに表示する文字列、*cancelButtonLabel* は “Cancel” ボタンの位置に “Cancel” の代わりに表示する文字列、*wanted* は返したい値のタイプで `string`、`unicode`、`AFSSpec`、`FSRef` とそれらのサブタイプを指定できます。

その他の引数の説明については Apple Navigation Services のドキュメントと EasyDialogs のソースコー

ドを参照してください。

AskFileForSave([*message*] [, *savedFileName*] [, *defaultLocation*] [, *defaultOptionFlags*] [, *location*] [, *clientName*] [, *windowTitle*] [, *actionButtonLabel*] [, *cancelButtonLabel*] [, *preferenceKey*] [, *popupExtension*] [, *fileType*] [, *fileCreator*] [, *eventProc*] [, *wanted*])

保存するファイルをユーザに尋ねるダイアログを表示して、選択されたファイルあるいはユーザがキャンセルしたら *None* を返します。*savedFileName* は保存するファイルのデフォルトの名前（であり、返される値）です。その他の引数の説明については AskFileForOpen を参照してください。

AskFolder([*message*] [, *defaultLocation*] [, *defaultOptionFlags*] [, *location*] [, *clientName*] [, *windowTitle*] [, *actionButtonLabel*] [, *cancelButtonLabel*] [, *preferenceKey*] [, *popupExtension*] [, *eventProc*] [, *filterProc*] [, *wanted*])

選択するフォルダをユーザに尋ねるダイアログを表示して、選択されたフォルダあるいはユーザがキャンセルしたら *None* を返します。引数についての説明は AskFileForOpen を参照してください。

2.8.1 プログレスバーオブジェクト

`ProgressBar` オブジェクトはモードレスプログレスバーの土台を提供します。定量プログレスインジケータ（温度計スタイル）と不定量プログレスインジケータ（床屋さんのぐるぐる看板のスタイル）が使用できます。プログレスバーは最大値がゼロより大きければ定量インジケータに、そうでないなら不定量インジケータになります。 2.2 で変更された仕様: 不定量プログレスインジケータが追加されました

ダイアログは作られるとすぐに表示されます。ダイアログの “Cancel” ボタンが押されるか、`Cmd-.` (コマンドキーを押しながらピリオド. を押す) か `ESC` がタイプされるとダイアログウィンドウは隠され、`KeyboardInterrupt` が発生します（しかし、この反応はプログレスバーが次にアップデートされるまで、つまり `inc()` か `set()` の呼び出しによって次にアップデートされるまで発生しません）。それ以外は、プログレスバーは `ProgressBar` オブジェクトが捨てられるまで表示されたままになります。

`ProgressBar` オブジェクトには以下の属性とメソッドがあります。

curval

プログレスバーの現在の値（整数型あるいはロング整数型）。プログレスバーの通常のアクセスのメソッドによって `curval` を `0` と `maxval` の間にします。この属性は直接変更すべきではありません。

maxval

プログレスバーの最大の値（整数型あるいはロング整数型。`curval` が `maxval` に等しい時にはプログレスバー（温度計スタイル）は最大値を示します。もし `maxval` が `0` なら、プログレスバーは不定量インジケータ（床屋さんのぐるぐる看板）です。この属性は直接変更すべきではありません。

title([*newstr*])

プログレスバーのダイアログのタイトルバーのテキストを *newstr* に設定します。

label([*newstr*])

プログレスバーのダイアログのプログレスボックスのテキストを *newstr* に設定します。

set(*value*[, *max*])

プログレスバーの現在値 `curval` を *value* に設定し、もし *max* が指定されたら `maxval` を *max* にします。*value* は初めに 0 と `maxval` の間に強制的に設定されます。温度計スタイルのバーは変化を反映して更新され、また不定量インジケータも定量インジケータへ、あるいはその逆も起こります。

inc([*n*])

プログレスバーの `curval` を *n* だけ増やし、*n* が指定されないなら `1` だけ増やします。（*n* は負でもかまいません。その場合は減少します。）プログレスバーは変化を反映してアップデートされます。もしプログレスバーが不定量インジケータなら、床屋さんのぐるぐる看板が 1 回 “スピン” します。もし増加された結果が 0 と `maxval` の範囲外になったら、`curval` は 0 と `maxval` の間に強制的に設

定されます。

2.9 FrameWork — 対話型アプリケーション・フレームワーク

FrameWork モジュールは、対話型 Macintosh アプリケーションのクラスで、同時にフレームワークを提供します。プログラマは、サブクラスを作って基底クラスの様々なメソッドをオーバーライドし、必要な機能を実装することでアプリケーションを組み立てられます。機能のオーバーライドは、時によって様々な異なるレベルで行われます。つまり、ある一つのダイアログウィンドウでクリックの処理を普段と違う方法で行うには、完全なイベント処理をオーバーライドする必要はありません。

FrameWork はまだ作成中のところがずいぶん残っています。このドキュメントでは最も重要な機能だけしか記述していませんし、それさえも論理的な形で書かれてもいません。ソースか例題を詳しく見てください。次にあげるのは、MacPython ニュースグループにポストされたコメントで、FrameWork の強力さと限界について述べています。

> FrameWork の最大の強みは、制御の流れをたくさんの異なる部分に分割できることです。例えば W を使って、いろいろな方法でメニューをオン/オフしたり、残りをいじらずにうまくプラグインさせることができます。FrameWork の弱点は、コマンドインタフェースが抽象化されていないこと (といっても難しいわけではないですが)、ダイアログサポートが最低限しかないこと、それからコントロール/ツールバーサポートが全くないことです。

FrameWork モジュールは次の関数を定義しています。

Application()
: アプリケーション全体を表現しているオブジェクト。メソッドについての詳細は以下の記述を参照してください。デフォルト __init__() ルーチンは、空のウィンドウ辞書とアップルメニューつきのメニューバーをを作成します。

MenuBar()
: メニューバーを表現するオブジェクト。このオブジェクトは普通はユーザは作成しません。

Menu(*bar, title*[, *after*])
: メニューを表現するオブジェクト。生成時には、メニューが現われる MenuBar と、*title* 文字列、メニューが表示されるべき (1 から始まる) 位置 *after* (デフォルトは末尾) を渡します。

MenuItem(*menu, title*[, *shortcut, callback*])
: メニューアイテムオブジェクトを作成します。引数は作成するメニューと、アイテムのタイトル文字列、オプションのキーボードショートカット、コールバックルーチンです。コールバックは、メニュー ID、メニュー内のアイテム番号 (1 から数える)、現在のフロントウィンドウ、イベントレコードを引数に呼ばれます。

  呼び出し可能なオブジェクトのかわりに、コールバックは文字列でも良いです。この場合、メニューの選択は、最前面のウィンドウとアプリケーションの中でメソッド探索を引き起こします。メソッド名は、コールバック文字列の前に'domenu_' を付けたものです。

  MenuBar の fixmenudimstate() メソッドを呼びだすと、現在のフロントウィンドウにもとづいて、適切なディム化を全てのメニューアイテムに対してほどこします。

Separator(*menu*)
: メニューの最後にセパレータを追加します。

SubMenu(*menu, label*)
: *label* の名前のサブメニューを、メニュー *menu* の下に作成します。メニューオブジェクトが返されます。

Window(*parent*)
(モードレス) ウィンドウを作成します。*Parent* は、ウィンドウが属するアプリケーションオブジェクトです。作成されたウィンドウはまだ表示されません。

DialogWindow(*parent*)
モードレスダイアログウィンドウを作成します。

windowbounds(*width, height*)
与えた幅と高さのウィンドウを作成するのに必要な、(*left*, *top*, *right*, *bottom*) からなるタプルを返します。ウィンドウは以前のウィンドウに対して位置をずらして作成され、全体のウィンドウが画面からなるべく外れないようにします。しかし、ウィンドウはいつでも全く同じサイズで、そのため一部は画面から隠れる場合もあります。

setwatchcursor()
マウスカーソルを時計型に設定します。

setarrowcursor()
マウスカーソルを矢印型に設定します。

2.9.1 アプリケーションオブジェクト

アプリケーションオブジェクトのメソッドは各種ありますが、次のメソッドをあげておきます。

makeusermenus()
アプリケーションでメニューを使う必要がある場合、このメソッドをオーバーライドします。属性 `menubar` にメニューを追加します。

getabouttext()
このメソッドをオーバーライドすることで、アプリケーションの説明を記述するテキスト文字列を返します。代わりに、`do_about()` メソッドをオーバーライドすれば、もっと凝った “アバウト” メッセージを出す事ができます。

mainloop([*mask*[, *wait*]])
このルーチンがメインイベントループで、作成したアプリケーションが動き出すためにはこれを呼ぶことになります。*Mask* は操作したいイベントを選択するマスクです。 *wait* は並行に動作しているアプリケーションに割り当てたいチック数 (1/60 秒) です (デフォルトで 0 ですが、あまり良い値ではありません)。*self* フラグを立ててメインループを抜ける方法はまだサポートされていますが、これはお勧めできません。代わりに `self._quit()` を呼んでください。

イベントループは小さなパーツに分割されていて、各々をオーバーライドできるようになっています。これらのメソッドは、デフォルトでウィンドウとダイアログや、ドラッグとリサイズの操作、AppleEvent、非 FrameWork のウィンドウに関するウィンドウの操作などに関するイベントを分岐することなどまで面倒をみてくれます。

原則として、全てのイベントハンドラは、イベントが完全に取り扱われた場合は `1` を返さなくてはいけませんし、それ以外では `0` を返さなくてはいけません (例えば、前面のウィンドウは FrameWork ウィンドウではない場合を考えてください)。こうしなくてはいけない理由は、アップデートイベントなどが Sioux コンソールウィンドウなどの他のウィンドウにきちんと渡されるようにするためです。*our_dispatch* やその呼び出し元の内部から `MacOS.HandleEvent()` を呼んではいけません。そうしたコードが Python の内部ループのイベントハンドラを経由して呼ばれると、無限ループになりかねないからです。

asyncevents(*onoff*)
非同期でイベント操作をしたい場合は、非ゼロの引数でこのメソッドを呼んでください。こうすることで、イベントが生じた時に、内部のインタプリタのループで、アプリケーションイベントハンドラ

async_dispatch が呼ばれることになります。すると、長時間の計算を行っている場合でも、FrameWork ウィンドウがアップデートされ、ユーザーインターフェースが動き続けるようになります。ただし、インタプリタの動作が減速し、非リエントラントのコード (例えば FrameWork 自身など) に奇妙な動作が見られるかもしれません。デフォルトでは *async_dispatch* はすぐに *our_dispatch* を呼びますが、このメソッドをオーバーライドすると、特定のイベントを非同期で操作しても良くなります。処理しないイベントは Sioux などに渡されることになります。

on あるいは off 値が返されます。

_quit()
: 実行中の `mainloop()` 呼び出しを、次の適当なタイミングで終了させます。

do_char(*c, event*)
: ユーザーが文字 *c* をタイプした時に呼ばれます。イベントの全詳細は event 構造体の中にあります。このメソッドはウィンドウオブジェクト内で使うためにも提供されています。このオブジェクトのウィンドウが最前面にある場合は、アプリケーション全般について本ハンドラをオーバーライドします。

do_dialogevent(*event*)
: イベントループ内部で最初に呼ばれて、モードレスダイアログイベントを処理します。デフォルトではメソッドは単にイベントを適切なダイアログに分岐するだけです (関連したダイアログウィンドウオブジェクトを経由してではありません)。特別にダイアログイベント (キーボードショートカットなど) を処理する必要がある場合にオーバーライドしてください。

idle(*event*)
: イベントが無い場合にメインイベントループから呼ばれます。 null イベントも渡されます (つまりマウス位置などを監視することができます)。

2.9.2 ウィンドウオブジェクト

ウィンドウオブジェクトは特に次のメソッドを持ちます。

open()
: ウィンドウを開く時はこのメソッドをオーバーライドします。MacOS ウィンドウ ID を `self.wid` に入れて `do_postopen()` メソッドを呼ぶと、親アプリケーションにウィンドウを登録します。

close()
: ウィンドウを閉じるときに特別な処理をする場合はこのメソッドをオーバーライドします。親アプリケーションからウィンドウの登録を削除するには、`do_postclose()` を呼びます。

do_postresize(*width, height, macoswindowid*)
: ウィンドウがリサイズされた後に呼ばれます。`InvalRect` を呼び出す以外にもすることがある場合はこれをオーバーライドします。

do_contentclick(*local, modifiers, event*)
: ウィンドウのコンテント部分をユーザーがクリックすると呼ばれます。引数は位置座標 (ウィンドウを基準)、キーモディファイア、生のイベントです。

do_update(*macoswindowid, event*)
: ウィンドウのアップデートイベントが受信された時に呼ばれます。ウィンドウを再描画します。

do_activate(*activate, event*)
: ウィンドウがアクティブ化 (*activate* `== 1`)、非アクティブ化 (*activate* `== 0`) する際に呼ばれます。フォーカスのハイライトなどを処理します。

2.9.3 コントロールウィンドウオブジェクト

コントロールウィンドウオブジェクトには `Window` オブジェクトのメソッドの他に次のメソッドがあります。

`do_controlhit`(*window, control, pcode, event*)
: コントロール *control* のパートコード *pcode* がユーザーにヒットされた場合に呼ばれます。トラッキングなどは任せておいてかまいません。

2.9.4 スクロールウィンドウオブジェクト

スクロールウィンドウオブジェクトは、次のメソッドを追加したコントロールウィンドウオブジェクトです。

`scrollbars`([*wantx*[, *wanty*]])
: 水平スクロールバーと垂直スクロールバーを作成します (あるいは破棄します)。引数はどちらが欲しいか指定します (デフォルトは両方)。スクロールバーは常に最小値 `0` 、最大値 `32767` です。

`getscrollbarvalues`()
: このメソッドは必ず作っておかなくてはいけません。現在のスクロールバーの位置を与えるタプル (*x, y*) を (`0` の `32767` 間で) 返してください。バーの方向について全文書が可視状態であること知らせるため `None` を返す事もできます。

`updatescrollbars`()
: 文書に変更があった場合はこのメソッドを呼びます。このメソッドは `getscrollbarvalues()` を呼んでスクロールバーを更新します。

`scrollbar_callback`(*which, what, value*)
: あらかじめ与えておくメソッドで、ユーザーとの対話により呼ばれます。*which* は `'x'` か `'y'` 、*what* は `'-'`, `'-'`, `'set'`,`'++'`, `'+'` のどれかです。`'set'` の場合は、*value* に新しいスクロールバー位置を入れておきます。

`scalebarvalues`(*absmin, absmax, curmin, curmax*)
: `getscrollbarvalues()` の結果から値を計算するのを助ける補助的なメソッドです。文書の最小値と最大値、可視部分に関する最先頭値 (最左値) と最底値 (最右値) を渡すと、正しい数か `None` を返します。

`do_activate`(*onoff, event*)
: ウィンドウが最前面になった時、スクロールバーのディム (dimming)/ハイライトの面倒をみます。このメソッドをオーバーライドするなら、オーバーライドしたメソッドの最後でオリジナルのメソッドを呼んでください。

`do_postresize`(*width, height, window*)
: スクロールバーを正しい位置に移動させます。オーバーライドする時は、オーバーライドしたメソッドの一番最初でオリジナルのメソッドを呼んでください。

`do_controlhit`(*window, control, pcode, event*)
: スクロールバーのインタラクションを処理します。これをオーバーライドする時は、オリジナルのメソッドを最初に呼び出してください。非ゼロの返り値はスクロールバー内がヒットされたことを意味し、実際に処理が進むことになります。

2.9.5 ダイアログウィンドウオブジェクト

ダイアログウィンドウオブジェクトには、`Window` オブジェクトのメソッドの他に次のメソッドがあります。

`open`(*resid*)

ID *resid* の DLOG リソースからダイアログウィンドウを作成します。ダイアログオブジェクトは `self.wid` に保存されます。

do_itemhit(*item, event*)

アイテム番号 *item* がヒットされた時に呼ばれます。トグルボタンなどの再描画は自分で処理してください。

2.10 autoGIL — イベントループ中のグローバルインタープリタの取り扱い

`autoGIL` モジュールは、自動的にイベントループを実行する場合 、Python のグローバルインタープリタをロックしたり、ロックの解除をしたりするための関数 `installAutoGIL` を提供します。

exception AutoGILError

例えば現在のスレッドがループしていないなど、オブザーバにコールバックができない場合に発生します。

installAutoGIL()

現在のスレッドのイベントループ (CFRunLoop) 中のオブザーバにコールバックを行ない、適切な時にグローバルインタープリタロック (GIL) を、イベントループが使用されていない間、他の Python スレッドの起動ができるようにロックしたり、ロックの解除をしたりします。

有効性：OSX 10.1 以降

三章

MacPython OSA モジュール

Python は オープンスクリプティングアーキテクチャ(Open Scripting Architecure、OSA、一般的には AppleScript と呼ばれる) のかなり完全な実装を行っていて、Python プログラムからスクリプト可能なアプリケーションを操作したり、Python へのインターフェースを備えたものにすることができます。

AppleScript と OSA の様々なコンポーネントの記述のために、また、アーキテクチャおよび用語についての理解を得るために、アップルの文書を読む必要があります。"Applescript Language Guide" は概念のモデルおよび用語、Standard Suite について説明した文書です。"Open Scripting Architecture" 文書は、アプリケーションプログラマの視点から OSA を使用する方法について説明しています。これらの文書は Apple ヘルプビューワの Developer Documentation 中の Core Technologies セクションにあります。

アプリケーションをスクリプトで操作する例として、次の AppleScript は、一番前の **Finder** ウィンドウの名前を取得し、それを印字します。

```
tell application "Finder"
    get name of window 1
end tell
```

Python では以下のコードで同じ事ができます。

```
import Finder

f = Finder.Finder()
print f.get(f.window(1).name)
```

配布されている Python ライブラリは、Standard Suite を実装したパッケージに加えて、いくつかの一般的なアプリケーションへのインターフェースを実装したパッケージが含まれています。

アプリケーションに AppleEvent を送るためには、アプリケーションの用語 (**Script Editor** が「辞書」と呼ぶもの) に接続する Python パッケージを最初に作成しなければなりません。これは、**PythonIDE** の内部から、あるいは、コマンドラインからのスタンドアロンのプログラムとして 'gensuitemodule.py' モジュールを実行することにより行うことができます。

'gensuitemodule.py' モジュールで生成される出力は多くのモジュールを備えたパッケージのため、全ての Suite をプログラムの中で 1 つにまとめて利用できるようにするために `__init__` モジュールが追加されています。Python 継承グラフは AppleScript 継承グラフを理解するので、Standard Suite をサポートしていて、余分な引数を備えた 1 つあるいは 2 つの変数を拡張する事ができるようにプログラム辞書が書かれていた場合、出力された Suite は、`StdSuites.Standard_Suite` からすべてをインポートして再エクスポートし、さらに拡張機能をもったメソッドをオーバーライドするモジュール `Standard_Suite` を含みます。`gensuitemodule` の出力は人間に判読可能で、Python docstrings 中にはオリジナルの AppleScript 辞書にあった文書を含んでいます。したがって、それを読むことは有用な情報源となります。

出力されたパッケージは、メソッドとして AppleScript 変数をすべて含み、第 1 の引数としての直接オブ

ジェクトを含み、キーワード引数としてのすべてのオプションの引数を含む、パッケージと同じ名前を備えた主要なクラスを実装しています。また AppleScript クラスは Python クラス、そして類事物その他のもろもろの物として実装されています。

変数を実装する主要な Python クラスは、さらに AppleScript クラス"application" で宣言されたプロパティおよび要素へのアクセスを許可します。現在のリリースでオブジェクト指向的にやろうとするならば、例えば、より Python 的な `f.window(1).name.get()` の代りに `f.get(f.window(1).name)` を利用する必要があります。

AppleScript 識別子が Python 識別子と同じでない場合、名前は少数の規則によって判別します。

- スペースは下線に置換されます。
- `_xx_` が 16 進法の文字値である場合、他の英数字でない文字は `xx` と置換されます。
- あらゆる Python 予約語には下線を追加します。

Python は、さらに Python でスクリプト対応アプリケーションを作成する事をサポートしています。次のモジュールは MacPython の AppleScript サポートに適切です。

`gensuitemodule`	OSA 辞書からスタブパッケージを作成します。
`aetools`	Apple Event を送るための基本的なサポート
`aepack`	Python 変数と AppleEvent データコンテナ間の変換
`aetypes`	Apple Event オブジェクトモデルの Python 表現
`MiniAEFrame`	オープンスクリプティングアーキテクチャ(OSA) サーバ (“Apple Events”) のサポート。

さらに、`Finder`, `Terminal`, `Explorer`, `Netscape`, `CodeWarrior`, `SystemEvents` そして `StdSuites` のサポートモジュールは、あらかじめ生成されています。

3.1 gensuitemodule — OSA スタブ作成パッケージ

`gensuitemodule` モジュールは AppleScript 辞書によって特定のアプリケーションに実装されている AppleScript 群のためのスタブコードを実装した Python パッケージを作成します。

このモジュールは、通常は **PythonIDE** からユーザによって起動されますが、コマンドラインからスクリプトとして実行する (オプションとしてヘルプに **--help** を与えてみてください) こともできますし、Python コードでインポートして利用する事もできます。使用例として、どのようにして標準ライブラリに含まれているスタブパッケージを生成するか、‘Mac/scripts/genallsuites.py’ にあるソースを見てください。

このモジュールは次の関数を定義しています。

`is_scriptable`(*application*)

`application` としてパス名を与えたアプリケーションがスクリプト可能でありそうな場合、真を返します。返り値はやや不確実な場合があります。**Internet Explorer** はスクリプト不可能なように見えてしまいますが、実際はスクリプト可能です。

`processfile`(*application*[, *output, basepkgname, edit_modnames, creatorsignature, dump, verbose*])

パス名として渡された `application` のためのスタブパッケージを作成します。‘.app’ として一つのパッケージにまとめてあるプログラム群のために内部の実行プログラムそのものではなくパッケージへのパス名を渡すだけでよくなっています。パッケージ化されていない CFM アプリケーションではアプリケーションバイナリのファイル名を渡す事もできます。

この関数は、アプリケーションの OSA 用語リソースを捜し、これらのリソースを読み取り、その結果データをクライアントスタブを実装した Python コードパッケージを作成するために使用します。

output は作成結果のパッケージを保存するパス名で、指定しない場合は標準の「別名で保存 (save file as)」ダイアログが表示されます。basepkgname はこのパッケージの基盤となるパッケージを指定します。デフォルトは StdSuites になります。StdSuites 自体を生成する場合だけ、このオプションを指定する必要があります。edit_modnames は自動生成によって作成されてあまり綺麗ではないモジュール名を変更するために使用することができる辞書です。creator_signature はパッケージ中の 'PkgInfo' ファイル、あるいは CFM ファイルクリエータ署名から通常得られる 4 文字クリエータコードを無視するために使用することができます。dump にはファイルオブジェクトを与えます、これを指定するとリソースを読取った後に停止して processfile がコード化した用語リソースの Python 表現をダンプします。verbose にもまたファイルオブジェクトを与え、これを指定すると processfile の行なっている処理の詳細を出力します。

processfile_fromresource(*application*[, *output, basepkgname, edit_modnames, creatorsignature, dump, verbose*])

この関数は、用語リソースを得るのに異なる方法を使用する以外は、processfile と同じです。この関数では、リソースファイルとして application を開き、このファイルから "aete" および "aeut" リソースをすべて読み込む事で、AppleScript 用語リソース読み込みを行ないます。

3.2 aetools — OSA クライアントのサポート

aetools モジュールは Python で AppleScript クライアントとしての機能をサポートするアプリケーションを構築するための基本的な機能を含んでいます。さらに、このモジュールは、 aetypes および aepack モジュールの中核機能をインポートし再 エクスポートします。gensuitemodule によって生成されたスタブパッケージは aetools のかなり適切な部分をインポートするので、通常はそれを明示的にインポートする必要はありません。生成されたパッケージ群を使用することができない場合と、スクリプト対応のためにより低いレベルのアクセスを必要としている場合、例外が発生します。

aetools モジュールはそれ自身、Carbon.AE モジュールによって提供される AppleEvent サポートを利用します。このモジュールにはウィンドウマネージャへのアクセスを必要とするという 1 つの欠点があります。詳細は第 1.1.2 章を見てください。この制限は将来のリリースで撤廃されるかもしれません。

aetools モジュールは下記の関数を定義しています。

packevent(*ae, parameters, attributes*)

あらかじめ作成された Carbon.AE.AEDesc オブジェクト中のパラメーターおよび属性を保存します。parameters と attributes は Python オブジェクトの 4 文字の OSA パラメータのキーを写像した辞書です。このオブジェクトをパックするには aepack.pack() を使います。

unpackevent(*ae*[, *formodulename*])

再帰的に、Carbon.AE.AEDesc イベントを Python オブジェクトへアンパックします。関数は引数の辞書および属性の辞書を返します。formodulename 引数は AppleScript クラスをどこに捜しに行くか制御するために、生成されたスタブパッケージにより使用されます。

keysubst(*arguments, keydict*)

Python キーワード引数辞書 arguments を、写像による 4 文字の OSA キーとして keydict の中で指定された Python 識別子であるキーの交換により packevent によって要求されるフォーマットへ変換します。生成されたパッケージ群によって使用されます。

enumsubst(*arguments, key, edict*)

arguments 辞書が key へのエントリーを含んでいる場合、辞書 edict のエントリーに見合う値に変換します。これは人間に判読可能なように Python 列挙名を OSA 4 文字のコードに変換します。生成されたパッケージ群によって使用されます。

aetools モジュールは次のクラスを定義しています。

class **TalkTo**([*signature=None, start=0, timeout=0*])

アプリケーションとの対話に利用する代理の基底クラスです。signature はクラス属性 _signature (サブクラスによって通常設定される) を上書きした、対話するアプリケーションを定義する 4 文字クリエートコードです。start にはクラスインスタンス上でアプリケーションを実行することを可能にするために、真を設定する事ができます。timeout を明示的に設定する事で、 AppleEvent の返答を待つデフォルトのタイムアウト時間を変更する事ができます。

_start()

アプリケーションが起動していてるか確認し、起動していなければ起動しようとします。

send(*code, subcode*[, *parameters, attributes*])

OSA 指示子 code, subcode (いずれも通常 4 文字の文字列です) を持った変数のために、parameters をパックし、attributes に戻し、目標アプリケーションにそれを送って、返答を待ち、unpackevent を含んだ返答をアンパックし、AppleEvent の返答を返し、辞書としてアンパックした値と属性を返して、AppleEvent Carbon.AE.AEDesc を作成します。

3.3 aepack — Python 変数と AppleEvent データコンテナ間の変換

aepack モジュールは、Python 変数から AppleEvent ディスクリプタへの変換 (パック) と、その逆に変換 (アンパック) する関数を定義しています。Python 内では AppleEvent ディスクリプタは、組み込み型である AEDesc の Python オブジェクトとして扱われます。AEDesc は Carbon.AE モジュールで定義されています。

aepack モジュールは次の関数を定義しています。

pack(*x*[, *forcetype*])

Python 値 x を変換した値を保持する AEDesc オブジェクトを返します。*forcetype* が与えることで、結果のディスクリプタ型を指定できます。それ以外では、Python 型から Apple Event ディスクリプタ型へのデフォルトのマッピングが使われます。マッピングは次の通りとなります。

Python type	descriptor type
FSSpec	typeFSS
FSRef	typeFSRef
Alias	typeAlias
integer	typeLong (32 bit integer)
float	typeFloat (64 bit floating point)
string	typeText
unicode	typeUnicodeText
list	typeAEList
dictionary	typeAERecord
instance	*see below*

x が Python インスタンスなら、この関数は __aepack__() メソッドを呼びだそうとします。このメソッドは AEDesc オブジェクトを返します。

x の変換が上で定義されていない場合は、この関数は、テキストディスクリプタとしてエンコードされた、値の (repr() 関数による)Python 文字列表現が返されます。

unpack(*x*[, *formodulename*])

x は AEDesc タイプのオブジェクトでなければいけません。この関数は、Apple Event ディスクリプタ *x* のデータの Python オブジェクト表現を返します。単純な AppleEvent データ型 (整数、テキスト、浮動少数点数) の、対応する Python 型が返されます。Apple Event リストは Python リストとして返され、リストの要素は再帰的にアンパックされます。formodulename の指定がない場合、オブジェ

クト参照 (例：`line 3 of document 1`) が、`aetypes.ObjectSpecifier` のインスタンスとして返されます。ディスクリプタ型が typeFSS である AppleEvent ディスクリプタが、`FSSpec` オブジェクトとして返されます。AppleEvent レコードディスクリプタが、再帰的にアンパックされた、型の 4 文字キーと要素を持つ Python 辞書として返されます。

オプションの `formodulename` 引数は `gensuitemodule` より作成されるスタブパッケージにより利用され、オブジェクト指定子のための OSA クラスをモジュールの中で見つけられることを保証します。これは、例えば、ファインダがウィンドウに対してオブジェクト指定子を返す場合、`Finder.Window` のインスタンスが得られ、`aetypes.Window` が得られないことを保証します。前者は、ファインダ上のウィンドウが持っている、すべての特性および要素のことを知っています。一方、後者のものはそれらのことを知りません。

参考資料:

`Carbon.AE` モジュール (4.1 節):
Apple Event マネージャルーチンへの組み込みアクセス

`aetypes` モジュール (3.4 節):
Apple Event ディスクリプタ型としてコードされた Python 定義

Inside Macintosh: Interapplication Communication
(`http://developer.apple.com/techpubs/mac/IAC/IAC-2.html`)
Macintosh 上でのプロセス間通信に関する情報

3.4 `aetypes` — AppleEvent オブジェクト

`aetypes` は Apple Event データディスクリプタおよび Apple Event オブジェクト指定子を表わすために使用されるクラスを定義します。

Apple Event データはディスクリプタに含まれていて、これらのディスクリプタは型です。多くのディスクリプタについては、その Python 表現は単に対応する Python の型になります。例えば、OSA 中の `typeText` は Python 文字列型で、`typeFloat` は 浮動小数点型、などのようになります。直接 Python で相当する型がない OSA の型については、このモジュールがクラスを宣言します。これらのクラスのインスタンスのパック、アンパックは `aepack` によって自動的に処理されます。

オブジェクト指定子は本質的に Apple Event サーバーの中で実行しているオブジェクトのアドレスです。Apple Event 指定子は、Apple Event のための直接オブジェクト、あるいはオプションパラメータの引数として使用されます。`aetypes` モジュールは OSA クラス、およびプロパティのための基底クラスを含んでいて、それらは、与えられたクラス群やプロパティ群のクラスおよびプロパティに含めるために、`gensuitemodule` によって生成されたパッケージによって使用されます。

下位互換性のためと、スクリプトにスタブパッケージを生成していないアプリケーションを必要とする場合のために、このモジュールはさらに `Document`、`Window`、`Character`、などのような多くの共通 OSA クラス用のオブジェクト指定子を含んでいます。

`AEObjects` モジュールは Apple Event ディスクリプタデータを表わすために次のクラスを定義しています。

class `Unknown`(*type, data*)
これ以外のクラスによって表わされず、単純な Python の値と等価でないような、`aepack` および `aetypes` モジュールでサポートしていない OSA ディスクリプタデータの表現。

class `Enum`(*enum*)
与えられた 4 文字のストリング値を持った列挙値。

class `InsertionLoc`(*of, pos*)

オブジェクト `of` の中の `pos` の位置。

class **`Boolean`**(*bool*)
真偽値。

class **`StyledText`**(*style, text*)
スタイル情報 (フォント、フェイスなど) を持っているテキスト。

class **`AEText`**(*script, style, text*)
スクリプトシステムおよびスタイル情報を持っているテキスト。

class **`IntlText`**(*script, language, text*)
スクリプトシステムおよび言語情報を持っているテキスト。

class **`IntlWritingCode`**(*script, language*)
スクリプトシステムと言語情報。

class **`QDPoint`**(*v, h*)
QuickDraw の点。

class **`QDRectangle`**(*v0, h0, v1, h1*)
QuickDraw の矩形。

class **`RGBColor`**(*r, g, b*)
色。

class **`Type`**(*type*)
4 文字の名前を持つ OSA 型の値。

class **`Keyword`**(*name*)
4 文字の名前をを持つ OSA のキーワード。

class **`Range`**(*start, stop*)
範囲。

class **`Ordinal`**(*abso*)
`"firs"` で最初とか、`"midd"` で中央、のような数値表現でない絶対位置。

class **`Logical`**(*logc, term*)
`term` にオペレーター `logc` を適用する論理表現。

class **`Comparison`**(*obj1, relo, obj2*)
`obj1` と `obj2` の `relo` による比較。

Python の中で AppleScript クラスおよびプロパティを表わすために、次のクラスは生成するスタブパッケージの基底クラスとして使用されます。

class **`ComponentItem`**(*which*[, *fr*])
OSA クラス用の抽象基底クラスです。サブクラスには 4 文字の OSA クラスコードのクラス属性 `want` を設定する必要があります。このクラスのサブクラスのインスタンスは AppleScript オブジェクト指定子と等価になります。インスタンス化する時、`which` の値としてセレクターを渡す事は必須で、`fr` 中の親オブジェクトを渡すかは任意です。

class **`NProperty`**(*fr*)
OSA プロパティ用の抽象基底クラスです。サブクラスはクラス属性として `want` と `which` を設定する必要がありあます。このクラスのサブクラスのインスタンスはオブジェクト指定子と等価です。

class **`ObjectSpecifier`**(*want, form, seld*[, *fr*])
`ComponentItem` と `NProperty` の基底クラスで、一般的な OSA オブジェクト指定子です。引数の内容は Apple オープンスクリプティングアーキテクチャの文書を御覧ください。このクラスは抽象クラスではない事に注意してください。

3.5 MiniAEFrame — オープンスクリプティングアーキテクチャサーバのサポート

MiniAEFrame モジュールは、アプリケーションにオープンスクリプティングアーキテクチャ(OSA) サーバ機能を持たせるためのフレームワークを提供します。つまり、AppleEvents の受信と処理を行わせます。FrameWork と連携させても良いし、単独でも使えます。実例として、このモジュールは **PythonCGISlave** の中で使われています。

MiniAEFrame には以下のクラスが定義されています。

class AEServer()

AppleEvent の分岐を処理するクラス。作成するアプリケーションはこのクラスと、MiniApplication あるいは FrameWork.Application のサブクラスでなければなりません。サブクラス化したクラスでは__init__() メソッドで、継承した両方のクラスの__init__() メソッドを呼びださなければなりません。

class MiniApplication()

FrameWork.Application とある程度互換なクラスですが、機能は少ないです。このクラスのイベントループはアップルメニュー、Cmd-.(コマンドキーを押しながらピリオド. を押す)、AppleEvent をサポートします。他のイベントは Python インタープリタか Sioux（CodeWarrior のコンソールシステム）に渡されます。作成するアプリケーションで AEServer を使いたいが、独自のウィンドウなどを持たない場合に便利です。

3.5.1 AEServer オブジェクト

installaehandler(*classe, type, callback*)

AppleEvent ハンドラをインストールします。*classe* と *type* は 4 文字の OSA クラスとタイプの指定子で、ワイルドカード'****' も使えます。対応する AppleEvent を受けるとパラメータがデコードされ、与えたコールバックが呼び出されます。

callback(*_object, **kwargs*)

与えたコールバックは、OSA ダイレクトオブジェクトを 1 番目のパラメータとして呼び出されます。他のパラメータは 4 文字の指定子を名前にしたキーワード引数として渡されます。他に 3 つのキーワード・パラメータが渡されます。つまり、_class と_type はクラスとタイプ指定子で、_attributes は AppleEvent 属性を持つ辞書です。

与えたメソッドの返り値は aetools.packevent() でパックされ、リプライとして送られます。

現在のクラス設計にはいくつか重大な問題があることに注意してください。引数に名前ではない 4 文字の指定子を持つ AppleEvent はまだ実装されていないし、イベントの送信側にエラーを返すこともできません。この問題は将来のリリースまで先送りにされています。

四章

MacOS ツールボックスモジュール

各種の MacOS ツールボックスへのインターフェースを与えるモジュール群があります。対応するモジュールがあるなら、そのモジュールではツールボックスで宣言された各種の構造体の Python オブジェクトが定義され、操作は定義されたオブジェクトのメソッドとして実装されています。その他の操作はモジュールの関数として実装されています。 C で可能な操作がすべて Python で可能なわけではありませんし (コールバックはよく問題になります)、パラメータが Python だと違ってしまうことはよくあります (特に入力バッファや出力バッファ)。全てのメソッドと関数は `__doc__` 文字列があるので、引数と返り値の説明を得る事ができます。他の情報源としては、*Inside Macintosh* などを参照してください。

これらのモジュールは全て `Carbon` パッケージに含まれています。この名前にもかかわらずそれら全てが Carbon フレームワークの一部なわけではありません。CF は、CoreFoundation フレームワークの中に実際はありますし、Qt は QuickTime フレームワークにあります。ツールボックスモジュールは普通以下のようにして利用します。

```
from Carbon import AE
```

注意！これらのモジュールはまだ文書化されていません。これらのモジュールのどれでもよいですが文書化に協力したいという方は、docs@python.org まで連絡をください。

Carbon.AE	Apple Event ツールボックスへのインタフェース
Carbon.AH	Apple ヘルプマネージャへのインタフェース
Carbon.App	アピアランスマネージャへのインタフェース
Carbon.CF	Core Foundation へのインタフェース
Carbon.CG	Component Manager へのインタフェース
Carbon.CaronEvt	Carbon Event Manager へのインタフェース
Carbon.Cm	Component Manager へのインタフェース
Carbon.Ctl	Control Manager へのインタフェース
Carbon.Dlg	Dialog Manager へのインタフェース
Carbon.Evt	Event Manager へのインタフェース
Carbon.Fm	Font Manager へのインタフェース
Carbon.Folder	Folder Manager へのインタフェース
Carbon.Help	Carbon Help Manager へのインタフェース
Carbon.List	List Manager へのインタフェース
Carbon.Menu	Menu Manager へのインタフェース
Carbon.Mlte	MultiLingual Text Editor へのインタフェース
Carbon.Qd	QuickDraw ツールボックスへのインタフェース
Carbon.Qdoffs	QuickDraw オフスクリーン API へのインタフェース
Carbon.Qt	QuickTime ツールボックスへのインタフェース
Carbon.Res	Resource Manager とハンドルへのインタフェース
Carbon.Scrap	Carbon Scrap Manager へのインタフェース
Carbon.Snd	Sound Manager へのインタフェース
Carbon.TE	TextEdit へのインタフェース
Carbon.Win	Window Manager へのインタフェース
ColorPicker	標準色選択ダイアログへのインターフェース

4.1 Carbon.AE — Apple Events

4.2 Carbon.AH — Apple ヘルプ

4.3 Carbon.App — アピアランスマネージャ

4.4 Carbon.CF — Core Foundation

CFBase, CFArray, CFData, CFDictionary, CFString と CFURL オブジェクトがいくらか部分的にサポートされています。

4.15 `Carbon.Menu` — Menu Manager

4.16 `Carbon.Mlte` — MultiLingual Text Editor

4.17 `Carbon.Qd` — QuickDraw

4.18 `Carbon.Qdoffs` — QuickDraw Offscreen

4.19 `Carbon.Qt` — QuickTime

4.20 `Carbon.Res` — Resource Manager and Handles

4.21 `Carbon.Scrap` — Scrap Manager

4.22 `Carbon.Snd` — Sound Manager

4.23 `Carbon.TE` — TextEdit

4.24 `Carbon.Win` — Window Manager

4.25 ColorPicker — 色選択ダイアログ

ColorPicker モジュールは標準色選択ダイアログへのアクセスを提供します。

GetColor(*prompt*, *rgb*)

標準色選択ダイアログを表示し、ユーザが色を選択することを可能にします。*prompt* の文字列によりユーザに指示を与えられ、デフォルトの選択色を *rgb* で設定する事ができます。*rgb* は赤、緑、青の色要素のタプルで与えてください。GetColor() はユーザが選択した色のタプルと色が選択されたか、取り消されたかを示すフラグを返します。

五章

文書化されていないモジュール

この章のモジュールは、(ここに記述があったとしても) 文書化がほとんどされていません。これらのモジュールのどれでもよいが文書化に協力したいという方は、docs@python.org まで連絡をください。

applesingle	AppleSingle フォーマットファイル用の基本的なデコーダ
buildtools	BuildApplet とその仲間のヘルパーモジュール
py_resource	コンパイル済みアプリケーションに 'PYC ' リソースを作成にするヘルパーモジュール
cfmfile	コードフラグメントリソースを扱うモジュール
icopen	open() と Internet Config の置き換え
macerrors	多くの MacOS エラーコード定数定義
macresource	スクリプトのリソースを見つける
Nac	Navigation Services へのインターフェース
mkcwproject	CodeWarrior プロジェクトの作成
nsremote	Netscape OSA モジュールのラッパー
PixMapWrapper	PixMap オブジェクトのラッパー
preferences	デフォルト設定へのサポートを持つアプリケーション初期設定管理プログラム
pythonprefs	Python インタープリタに特化した初期設定管理プログラム
quietconsole	バッファを用いての不可視の標準出力
videoreader	フレームの継続処理のための QuickTime ムービーのフレーム読み込み
W	FrameWork 上に作られた Mac 用ウイジェット
waste	“WorldScript-Aware Styled Text Engine” へのインターフェース

5.1 applesingle — AppleSingle デコーダー

5.2 buildtools — BuildApplet とその仲間のヘルパーモジュール

5.3 py_resource — Python コードからのリソース生成

このモジュールは **BuildApplet** と **BuildApplication** のヘルパーモジュールとして主に利用されています。コンパイル済みの Python コードに 'PYC ' リソースを付加する事ができます。

5.4 cfmfile — コードフラグメントリソースを扱うモジュール

cfmfile は、コードフラグメントと関連する “cfrg” リソースを処理するモジュールです。このモジュールでコードフラグメントを分解やマージできて、全てのプラグインモジュールをまとめて、一つの実行可能ファイルにするため、BuildApplication によって利用されます。

5.5 icopen — open() と Internet Config の置き換え

icopen をインポートすると、組込み open() を新しいファイル用にファイルタイプおよびクリエーターを設定するために Internet Config を使用するバージョンに置き換えます。

5.6 macerrors — MacOS のエラー

macerrors は、MacOS エラーコードを意味する定数定義を含みます。

5.7 macresource — スクリプトのリソースを見つける

macresource はスクリプトが MacPython 上や MacPython アプレットおよび OSX Python 上で起動されている時、特別な処理をせずにダイアログやメニューなどのようなリソースを見つけるためのヘルパースクリプトです。

5.8 Nav — NavServices の呼出し

Navigation Services の低レベルインターフェース。

5.9 mkcwproject — CodeWarrior プロジェクトの作成

mkcwproject は Metrowerks CodeWarrior 開発環境用のプロジェクトファイルを作成します。これは distutils のためのヘルパーモジュールですが、より多くの制御のために独立して利用できます。

5.10 nsremote — Netscape OSA モジュールのラッパー

nsremote は Netscape の OSA モジュールのラッパーであり、好きなブラウザに対して URL を簡単に送ることができます。*Python Library Reference* に記述のある webbrowser モジュールと緊密な関係があります。

5.11 PixMapWrapper — PixMap オブジェクトのラッパー

PixMapWrapper は PixMap オブジェクトを Python オブジェクトでラップしたもので、各フィールドに対し名前でアクセスできるようになります。PIL 画像との相互の変換をするメソッドも用意されています。

5.12 preferences — アプリケーション初期設定管理プログラム

preferences はシステム全体に渡る初期設定フォルダ中のユーザ毎の初期設定を記憶し、特定の状況に対する好みにより、デフォルトの設定を適用もしくは無視する事ができます。

5.13 pythonprefs — Python の初期設定管理プログラム

このモジュールは preferences モジュールをを特化させて Python インタープリタの初期設定を読み込んだり、書き込みしたりできるようにした物です。

5.14 quietconsole — 不可視の標準出力

quietconsole を使うと、バッファの stdio 出力を表示せずに (あるいは、**EditPythonPrefs** で設定されていれば、stdout ウィンドウを全く表示しないで) 保存することができます。保存されるのは、 stdin から読み込みを始めるか、バッファリングを止めるかするまでの間で、その時点で全ての出力は今度はウィンドウに送られるることになります。 グラフィカルユーザインターフェース (GUI) プログラムで、クラッシュ時の出力を表示したい場合に便利です。

5.15 videoreader — QuickTime ムービーの読み込み

videoreader は QuickTime ムービーを読み込み、デコードし、プログラムへ渡す事ができます。このモジュールはさらにオーディオトラックをサポートしています。

5.16 W — FrameWork 上に作られたウイジェット

W ウィジェットは、**IDE** で頻繁に使われています。

5.17 waste — Apple 製ではない **TextEdit** の置き換え

参考資料:

About WASTE
(http://www.merzwaren.com/waste/)
: WASTE ウィジェットとライブラリに関する情報サイト。ドキュメントとダウンロードもここから行なえます。

歴史とライセンス

A.1 History of the software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see http://www.cwi.nl/) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see http://www.cnri.reston.va.us/) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation; see http://www.zope.com/). In 2001, the Python Software Foundation (PSF, see http://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see http://www.opensource.org/ for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

Release	Derived from	Year	Owner	GPL compatible?
0.9.0 thru 1.2	n/a	1991-1995	CWI	yes
1.3 thru 1.5.2	1.2	1995-1999	CNRI	yes
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	no
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	yes
2.1.1	2.1+2.0.1	2001	PSF	yes
2.2	2.1.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2.1	2.2	2002	PSF	yes
2.2.2	2.2.1	2002	PSF	yes
2.2.3	2.2.2	2002-2003	PSF	yes
2.3	2.2.2	2002-2003	PSF	yes
2.3.1	2.3	2002-2003	PSF	yes
2.3.2	2.3.1	2003	PSF	yes

注意: GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike

the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

A.2 Terms and conditions for accessing or otherwise using Python

PSF LICENSE AGREEMENT FOR PYTHON 2.3.3

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 2.3.3 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 2.3.3 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001-2003 Python Software Foundation; All Rights Reserved" are retained in Python 2.3.3 alone or in any derivative version prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 2.3.3 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 2.3.3.

4. PSF is making Python 2.3.3 available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 2.3.3 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 2.3.3 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 2.3.3, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By copying, installing or otherwise using Python 2.3.3, Licensee agrees to be bound by the terms and conditions of this License Agreement.

BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0
BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization ("Licensee") accessing and otherwise using this software in source or binary form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement, BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use the Software alone or in any derivative version, provided, however, that the BeOpen Python License is retained in the Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis. BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects by the law of the State of California, excluding conflict of law provisions. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between BeOpen and Licensee. This License Agreement does not grant permission to use BeOpen trademarks or trade names in a trademark sense to endorse or promote products or services of Licensee, or any third party. As an exception, the "BeOpen Python" logos available at http://www.pythonlabs.com/logos.html may be used according to the permissions granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be bound by the terms and conditions of this License Agreement.

CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

1. This LICENSE AGREEMENT is between the Corporation for National Research Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191 ("CNRI"), and the Individual or Organization ("Licensee") accessing and otherwise using Python 1.6.1 software in source or binary form and its associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python 1.6.1 alone or in any derivative version, provided, however, that CNRI's License Agreement and CNRI's notice of copyright, i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version prepared by Licensee. Alternately, in lieu of CNRI's License Agreement, Licensee may substitute the following text (omitting the quotes): "Python 1.6.1 is made available subject to the terms and conditions in CNRI's License Agreement. This Agreement together with Python 1.6.1 may be located on the Internet using the following unique, persistent identifier (known as a handle): 1895.22/1013. This Agreement may also be obtained from a proxy server on the Internet using the following URL: `http://hdl.handle.net/1895.22/1013`."

3. In the event Licensee prepares a derivative work that is based on or incorporates Python 1.6.1 or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis. CNRI MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property law of the United States, including without limitation the federal copyright law, and, to the extent such U.S. federal law does not apply, by the law of the Commonwealth of Virginia, excluding Virginia's conflict of law provisions. Notwithstanding the foregoing, with regard to derivative works based on Python 1.6.1 that incorporate non-separable material that was previously distributed under the GNU General Public License (GPL), the law of the Commonwealth of Virginia shall govern this License Agreement only as to issues arising under or with respect to Paragraphs 4, 5, and 7 of this License Agreement. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between CNRI and Licensee. This License Agreement does not grant permission to use CNRI trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and conditions of this License Agreement.

ACCEPT

CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. All rights reserved.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH CENTRUM BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

日本語訳について

B.1 このドキュメントについて

この文書は、Python ドキュメント翻訳プロジェクトによる Macintosh Library Modules Release の日本語訳版です。日本語訳に対する質問や提案などがありましたら、Python ドキュメント翻訳プロジェクトのメーリングリスト

http://www.python.jp/mailman/listinfo/python-doc-jp

または、プロジェクトのバグ管理ページ

http://sourceforge.jp/tracker/?atid=116\&group_id=11\&func=browse

までご報告ください。

B.2 翻訳者一覧 (敬称略)

2.0 和訳: osawa <osawa at sm.rim.or.jp> (January, 2001)
2.3 差分和訳：sakito <sakito at s2.xrea.com>, Hiroyuki Yoshimura <DQB00103 at nifty.ne.jp> (October 10, 2003)
2.3.3 差分: Yasushi Masuda <y.masuda at acm.org>

MODULE INDEX

V

W

INDEX

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W